Symmetric and economical solutions to the mutual exclusion problem
in a distributed system

(Extended Abstract)

by

Shimon Cohen¹, Daniel Lehmann¹ and Amir Pnueli²

Abstract:

The mutual exclusion problem in a distributed system, in which each process has a memory of its own, into which it has exclusive write privileges but from which others may read, is reconsidered. Symmetric solutions are looked for. It is shown that, though no such solution may be deterministic, there are probabilistic solutions. Different solutions are provided for two processes, and then a solution is proposed for any number of processes. The solutions offered are amenable to a formal proof of their correctness with a small effort. The solutions are correct even against a very well informed scheduler, unlike Rabin's probabilistic solution to the mutual exclusion problem in a centralized system. Some of the solutions are correct even against an evil scheduler that knows in advance the results of the future random draws, in sharp contrast with the algorithms of [LR]. The solutions are economical: mutual exclusion between two processes may be achieved with variables capable of holding four different values (to be compared with Peterson and Fischer's three), mutual exclusion between n processes may be achieved with variables capable of holding ten different values (to be compared with Peterson and Fischer's fourteen). All solutions have been attained by careful reasoning and not by an exhaustive computer search, they exhibit general principles of design that may be useful in solving other similar problems.

1. The mutual exclusion problem in a distributed environment

The mutual exclusion problem is a now classical problem in concurrent programming, first proposed by E. W. Dijkstra (see [D1], [D2] and [K] for early work and [LA], [RP], [PF] and [R2] for recent work on this problem). For the notions of critical section, remainder, trying and exit sections the reader is referred to the papers quoted above.

Relative to the mutual exclusion problem, we define a lockout as a computation in which one of the processes wishes to enter its critical section, but will never do so. A deadlock is a computation in which, at some point, some process wishes to enter its critical section and no process ever enters its critical section beyond that point. A computation is said to exhibit

¹Institute of Mathematics and Computer Science, Hebrew University, Jerusalem 9104 (Israel)
²Department of Applied Mathematics, Weizmann Institute, Rehovoth (Israel)
overtaking bounded by \( k \). If every process that wishes, at any time, to enter its critical section, gets access to its critical section before any other process gets to enter its critical section \( k + 1 \) times.

We are interested in solving the mutual exclusion problem in a distributed environment as introduced in [LA]. We assume the existence of \( n \) processors, each containing its own memory unit. In each of those memory units, there is a special area that may be read (but not written) by any processor. Except for this special area, a processor has exclusive access to its own memory. Deterministic solutions to the mutual exclusion problem in a distributed environment have been proposed by [LA], [RF] and [PF]. The quality of a proposed solution is assessed by reference to different criteria, among them the size of the special area of memory used by the processors for communicating, the speed with which an interested processor will be allowed to enter its critical section and the immunity of the system to the possible failure of a processor. We concentrate on the first two criteria. The best solution proposed so far is that of [PF], where a solution is proposed for two processors that requires a special memory area capable of holding three different values (this number is also shown to be a lower bound), and a solution for \( n \) processors requiring an area capable of holding fourteen values. The solution also guarantees bounded waiting time.

2. Symmetric solutions

The solutions mentioned above are not symmetric, i.e. either the different processors follow different routines or the initial values of the memories of the different processors are not the same. Nevertheless one expects a solution not to favour one of the processors among its competitors. This requirement of symmetry has been first formulated by Dijkstra and an up-to-date study may be found in [BU].

A very simple symmetry argument can show that many problems do not have a deterministic symmetric solution, see for example [LR] and [LY]. The centralized version of the mutual exclusion problem has a symmetric deterministic solution. The distributed version does not.

There is wide agreement as to the necessity of behavioural symmetry, but not quite, yet, general agreement about the use of totally symmetric solutions. Symmetry is aesthetically pleasing (and that is important), but even more important is the fact that symmetric solutions automatically ensure behavioural symmetry (thus short-cutting a possibly delicate proof) and that their proofs of correctness tend to be eased by the symmetry. Symmetric solutions should also be preferred for economy reasons. A non-symmetric solution to the mutual exclusion problem for \( n \) processes, such as that of [PF], though it requires a shared (for reading only) variable of only constant (independent of \( n \)) size, requires each process to somehow remember some kind of identity number of size \( \log(n) \). A symmetric solution, such as ours, does not. Also, it is always easier to manufacture a system consisting of identical parts, than a system consisting of a large number of different parts that have to be assembled in a specific fixed layout. The rest of the paper is devoted to studying probabilistic symmetric solutions.

3. Probabilistic solutions

The main notions concerning probabilistic solutions, such as schedule, may be found in [LR]. The basic idea of all our solutions is to let all processes compete for the shared resource by drawing a random value and let the process that obtained the "highest" random value enter its critical section. The losers of a competition will then compete between themselves. If there is a tie the processes go through another competition. The two main problems that arise in implementing this idea are: to make sure that processes compare up-to-date results of random draws and not out-of-date values and to define precisely the group of processes competing, so as not to wait for results of draws of processes that are not interested in competing.